

|  |  |
| --- | --- |
| **ASSESSMENT TASK NOTIFICATION**  **(This sheet must be handed in with your task)** | |
| **Student:** | **Task Number:** 1 |
| **Subject:** Stage 6 – Software Engineering | **Weighting:** 25% |
| **Year / Class:** 12 HSC Course | **Date Issued to Class:** Term 4 – Week 2 |
| **Teacher:** D Hillman | **Date Due:** Term 4 – Week 6 Friday at 11:59PM |
| **Task Title:** CASE Tool Prototype | **Date Submitted:** |

|  |
| --- |
| **OUTCOMES ASSESSED:**  **SE-12-01**  justifies methods used to plan, develop and engineer software solutions  **SE-12-02**  applies structural elements to develop programming code  **SE-12-03**  analyses how current hardware, software and emerging technologies influence the development of software engineering solutions  **SE-12-06**  justifies the selection and use of tools and resources to design, develop, manage and evaluate software**SE-SE-12-08**  tests and evaluates language structures to refine code  **SE-12-09**  applies methods to manage and document the development of a software project |
| **TASK DESCRIPTION:**  In this task, you will select a CASE tool (e.g., Godot) to create a software prototype over a period of 5 weeks. The prototype can be part of a game, website app, or software application. The programming language must be Python or one with similar syntax. The final prototype must reflect effort over 5 weeks and include documentation following design thinking. Additionally, you must submit an evaluation of your process and product.  **TASK REQUIREMENTS:**  **Project Brief**  Your project brief outlines the requirements and details of your intended project. In your brief, include:   * Overview of the solution you plan on creating (1 paragraph)  |  | | --- | | I plan to create a first-person shooter (FPS) 3D game that combines strategic gameplay with immersive environments. The game will feature a dynamic, interactive world where players navigate through various levels, facing enemies, completing objectives, and overcoming obstacles. The focus will be on smooth controls, engaging combat mechanics, and visually appealing graphics that enhance the gaming experience. Developed using the Godot engine and programmed in GDScript (similar to Python), the prototype will showcase core gameplay features, including player movement, shooting mechanics, AI behaviour, and basic level design. The project will be documented through the design thinking process, ensuring a user-centered approach that iteratively refines the gameplay experience. |  * Detailed written features of the solution you plan on developing (1-2 paragraph)      |  | | --- | | The FPS game prototype will feature a core set of mechanics designed to provide an engaging and immersive experience. Key gameplay elements will include fluid player movement with sprinting, jumping, and crouching capabilities, as well as precise shooting mechanics with various weapon types. Players will be able to aim and fire accurately, with weapons featuring distinct properties such as rate of fire, accuracy, and damage. The game will also have an intuitive user interface displaying essential information like health, ammunition, and mission objectives.  To enhance gameplay, the prototype will include basic enemy AI that reacts to the player’s actions, providing challenging combat scenarios. Enemies will patrol specific areas, detect players within a certain range, and engage in combat when necessary. Level design will focus on creating an interactive environment with obstacles, cover spots, and tactical elements that encourage strategic gameplay. The game will also feature atmospheric sound effects and simple visual effects to make combat more engaging. Throughout the development, the design thinking process will be used to refine these features based on testing and feedback, ensuring the prototype is user-focused and offers a cohesive gaming experience. |  * Potential limitations you may encounter (1 paragraph)  |  | | --- | | One potential limitation in developing the FPS game prototype is the complexity of creating realistic and responsive enemy AI. Implementing behaviours that feel natural, such as patrolling, reacting to player actions, and using cover, can be challenging and may require significant time and testing to refine. Additionally, optimizing the game's performance, especially when rendering 3D graphics and handling real-time interactions, could be difficult given the processing demands. Another limitation is the scope of level design; creating diverse, visually appealing environments might be constrained by the limited development time of 5 weeks. Lastly, balancing gameplay elements, such as weapon mechanics and enemy difficulty, may prove challenging without thorough playtesting and feedback, which could be limited within the project's timeframe. |  * How will you overcome these limitations (1 paragraph)  |  | | --- | | To overcome these limitations, I will prioritize a modular approach to development, focusing on creating a solid foundation for the core gameplay mechanics before expanding features. For enemy AI, I will start with basic behaviours and iteratively improve them, ensuring they function correctly before adding more complex interactions. To address performance optimization, I will use efficient coding practices and regularly test the game on different devices to identify and resolve any issues early on. In terms of level design, I will create smaller, modular environments that can be easily adjusted or expanded, allowing for quicker iteration and flexibility. To ensure gameplay balance, I will conduct regular playtesting throughout the development process, gathering feedback and making adjustments as needed to create a smooth and engaging player experience. |   (**NOTE:** This is for what you intend to develop in the future. The project in its entirety. Not the prototype.)  **CASE Tool Research**  Conduct research on 5 different CASE tools that may be able to assist you with the development of this solution that you had ideated in your project brief. Include:   * CASE tool name (title)  |  | | --- | | Godot Engine |  * Description of the CASE tool (1 paragraph)  |  | | --- | | Godot Engine is a versatile, open-source game development platform that provides robust tools for creating 2D and 3D games. It supports a wide range of programming languages, with GDScript being its primary language, which has a syntax similar to Python. Godot offers a comprehensive suite of features, including a powerful scene system, visual scripting, built-in physics, and an intuitive user interface for designing and organizing game assets. Its modular design allows developers to create reusable components, making it easier to prototype and iterate on game ideas. With extensive support for scripting, visual effects, and cross-platform deployment, Godot is well-suited for rapid game development, especially for developers seeking a flexible and efficient workflow. |  * Advantages of using this CASE tool for the development of your solution (1 paragraph)  |  | | --- | | Using Godot Engine for the development of the FPS game prototype offers several advantages. Its intuitive and user-friendly interface makes it easy to quickly prototype and iterate on gameplay mechanics, which is essential given the 5-week development timeframe. The built-in scene and node system allows for modular development, enabling efficient organization of game assets and behaviours. GDScript, with its Python-like syntax, is easy to learn and implement, facilitating faster coding and debugging. Additionally, Godot’s integrated tools for handling 3D graphics, physics, and animation streamline the development of visually immersive and interactive environments. Being open-source, Godot is also cost-effective and has a large community, providing access to extensive resources, tutorials, and support that can aid in overcoming technical challenges during development. |  * Disadvantages of using this CASE tool for the development of your solution (1 paragraph)  |  | | --- | | Despite its advantages, using Godot Engine for the FPS game prototype comes with some disadvantages. While it is capable of handling 3D graphics, Godot is generally considered less powerful than other engines like Unity or Unreal when it comes to advanced 3D rendering and high-end visual effects. This could limit the level of graphical detail and realism achievable within the game. Additionally, the engine’s 3D physics and performance optimization tools, while functional, may require more manual tweaking to ensure smooth gameplay, which can be time-consuming. GDScript, though easy to use, is not as widely adopted as other languages like C# or C++, which means finding third-party libraries or plugins may be more challenging. Finally, the relatively smaller community of developers focusing on 3D projects in Godot might mean fewer resources and examples for solving specific 3D development issues. |  * Your opinion/perspective of the case tool. (1 paragraph)  |  | | --- | | In my opinion, Godot Engine is an excellent choice for this project because of its balance between simplicity and capability. I appreciate how intuitive the interface is, making it easy to organize and develop different elements of the game without getting bogged down in overly complex menus. The flexibility of GDScript allows me to write and test code quickly, which is a huge advantage given the limited time frame for the project. While Godot may not match the graphical power of some other engines, I believe its straightforward design and extensive built-in tools will allow me to focus on core gameplay elements without getting overwhelmed. I also like that it's open source, as this gives me more control over customization and helps keep the project cost-effective. |   **CASE Tool Selection and Justification**  Choose one of the five CASE tools you had conducted research to develop your prototype with.  Justify why this CASE tool was selected.  **Prototype Brief**  Your prototype brief outlines the requirements and details of your intended prototype. It is more specific. The selected CASE tool would typically inform the prototype you intend to develop (hence why it is shown AFTER the CASE Tool Selection and Justification documentation. Your prototype brief will include:   * Overview of the prototype you plan on creating (1 paragraph)  |  | | --- | |  |  * Detailed written features of the prototype you plan on developing (1-2 paragraph)  |  | | --- | |  |  * Potential limitations you may encounter (1 paragraph)  |  | | --- | |  |  * How will you overcome these limitations (1 paragraph)  |  | | --- | |  |  Project Management Tasks This process still requires some of the fundamental project management documentation in order to ensure you manage your time and communicate your intentions with your client (your teacher).  **Logbook**  The purpose of the logbook is to keep a record of the time spent working on your project. Your logbook should include all the work and tasks undertaken to solving the problem. This includes tasks in your report, problems you encountered, solutions you found, and any research that you’ve done to help you. Include links and references to any resources that you’ve used to help you complete the tasks (websites, videos, etc.).   |  |  |  | | --- | --- | --- | | **Date** | **Tasks done**  (Tasks done,Problems encountered/ Solution found, research completed /needed) | **Resources used/needed** | |  |  |  | |  |  |  | |  |  |  | |  |  |  | |  |  |  | |  |  |  | |  |  |  | |  |  |  |   **Gantt Chart**  The Gantt chart is used to illustrate the different tasks you must complete and the time frame in which to complete them. Your Gantt chart needs to include your projected (intended) timeline of when to complete the tasks and the actual (realised) timeline across the project’s duration. Use the tasks in the report stages for your Gantt chart, adding sub-tasks where needed.    **A screenshot of a calendar  Description automatically generated**  **Development & Integration**  **Prototype Development**  Code and develop your prototype in your chosen CASE tool or with the Python programming language.   * 1. Consider the different user input data and the expected output.   2. Include comments in your code that describes what your code is doing.   3. Ensure that you are writing using the correct syntax and appropriate naming case conventions.   4. Include error handling techniques where applicable.   The primary focus of this task is to utilise your programming and planning skills. For this reason, aesthetics will not be assessed. However, you should still have acknowledgement of where elements for your program such as components of the GUI will be placed.  Refer to the marking criteria for a detailed breakdown of requirements.  **Github Commits**  Regular commits with detailed descriptions are completed within Github.  Appropriate version numbers are provided during each commit.  Github is used for yourself to revert corrupted copies of your project and by your teacher to track your progress.  **Evaluation**  Critically evaluate your prototype. In your evaluation, include:   * Reflection/recap on the project * Was your prototype a success or failure, why/why not * How you overcame the limitations of your project * Strengths and weaknesses of the project * Problems that you’ve encountered and how you resolved them * Would you continue to use this CASE tool or the prototype in the future? Why or why not?  |  | | --- | | ***\*\*\*If you fail to hand in this task on the due date you will receive a zero mark and an ‘N’ Award warning for non-completion of an assessment task will be issued.\*\*\**** | |
|  |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Marking Rubric** | | | | | |
|  | | | | | |
| **Project Ideation – 50 marks** | | | | | |
| **Criteria** | **Extensive** | **Thorough** | **Sound** | **Basic** | **Elementary** |
| **Project Brief**  **(10 marks)** | **(9 – 10 marks)**  The project brief provides a comprehensive and clear overview of the proposed solution in a concise paragraph, demonstrating a thorough understanding of the project requirements. The features of the proposed solution are detailed across 1-2 paragraphs, clearly outlining the functionalities, components, and distinctive characteristics. The brief also includes a well-explained paragraph on potential limitations, identifying foreseeable challenges and obstacles. In addition, a separate paragraph outlines effective strategies to overcome these limitations, showing creativity and resourcefulness. | **(7 – 8 marks)**  The project brief offers a clear overview of the proposed solution with only minor ambiguities. The features are described in detail, but some elements might require additional explanation or are less distinct. Potential limitations are addressed, though not exhaustively, with some gaps in identifying all possible challenges. The paragraph on overcoming limitations is clear and generally well-considered but might lack innovation or thoroughness. | **(5 – 6 marks)**  The project brief is generally adequate, with a reasonable overview of the proposed solution, though some aspects may be vague or require more clarity. The features are outlined but might miss some crucial details or need additional paragraphs to be comprehensive. The potential limitations section identifies some challenges but overlooks significant issues. The strategies to overcome limitations are outlined but are basic or lack depth. | **(3 – 4 marks)**  The project brief offers some overview of the proposed solution, but it is incomplete or unclear in several areas. The detailed features are insufficiently described, with significant gaps or a lack of clarity. The section on potential limitations has a basic mention of a few challenges, but it's lacking in detail or relevance. The paragraph on overcoming limitations is minimal and provides little insight into resolving the challenges. | **(1 – 2 marks)**  The project brief lacks a clear overview of the proposed solution, with significant confusion or misinterpretation. The features are either missing or described in a very elementary manner, with no clear indication of the key components or functionalities. The potential limitations section is largely absent or does not provide any meaningful insight into possible challenges. The strategies to overcome limitations are either not addressed or poorly considered, with no practical solutions presented. |
| **CASE Tool Research**  **(25 marks)** | **(21 – 25 marks)**  The research on 5 CASE tools is comprehensive, providing detailed descriptions, clear advantages, and disadvantages related specifically to the student’s project. The student’s opinion is well-justified, demonstrating deep critical thinking and an understanding of each tool’s technical capabilities and limitations. The connection to the project is strong, showing how each tool will assist in the development. The work is highly organised with clear headings, flawless grammar, and professional presentation. | **(16 – 20 marks)**  Research on 5 CASE tools is well-developed, with good descriptions of advantages and disadvantages. Some points could be expanded, but the overall analysis is thoughtful. The student’s opinion shows critical thinking, though reasoning could be more robust. The connection to the project is generally clear, and the work is well-organised with minor errors in presentation or clarity. | **(11 – 15 marks)**  Research on 5 CASE tools is adequate but lacks depth. Descriptions and analysis are present but may be vague or generalised, and the opinion is basic. The connection between the tools and the project is clear but lacks depth. The work is organised, but some clarity or structure is lacking, and there are several spelling or grammatical errors. | **(6 – 10 marks)**  Research on 3-4 CASE tools is present but lacks detail. Descriptions are vague, and the analysis is incomplete or unclear. The student’s opinion may be underdeveloped or simplistic, with weak connections to the project. The organisation is weak, and the work is marked by grammatical or structural errors. | **(1 – 5 marks)**  Research is either missing or incomplete, covering fewer than 3 tools or providing vague, irrelevant descriptions. The analysis of advantages and disadvantages is unclear, and the student’s opinion lacks any meaningful thought. Connections to the project are absent or superficial, and the work is poorly organised with significant grammatical or spelling errors. |
| **CASE Tool Selection and Justification**  **(5 marks)** | **(5 marks)**  The selection of the CASE tool is excellently justified with a clear and detailed explanation that connects the tool’s features to the project requirements. The justification demonstrates deep insight into how the tool’s advantages align with the specific needs of the prototype, showing thorough consideration of alternatives and explaining why other tools were not selected. The reasoning is critical, reflective, and demonstrates strong technical understanding. The explanation is well-organised, concise, and free of errors. | **(4 marks)**  The selected CASE tool is justified well with good reasoning, though some elements could be expanded. The connection between the tool’s features and the project is clear, though minor gaps in explanation may be present. Some comparison with alternative tools is offered, but it may lack depth. The reasoning shows technical understanding, though some aspects could be more developed. The explanation is generally well-organised with minor errors in clarity or structure. | **(3 marks)**  The selection of the CASE tool is adequately justified, though the reasoning may be more general or lacking in-depth analysis. The connection to the project’s needs is present but may not be fully explored. Comparison with alternative tools is minimal or missing. The justification shows some technical understanding, but it may be basic or lacking depth. The explanation is organised but may contain several errors or lack clarity. | **(2 marks)**  The CASE tool selection is justified, but the reasoning is weak or underdeveloped. The connection between the tool’s features and the project is unclear or vague. Little to no comparison is made with other tools, and the justification shows limited technical understanding. The explanation is poorly organised and contains significant errors in clarity or structure. | **(1 mark)**  The selection of the CASE tool is either missing or unjustified. There is little to no reasoning provided for why the tool was chosen, and no clear connection to the project’s requirements. The explanation is disorganised, unclear, and filled with grammatical or structural errors. |
| **Prototype Brief**  **(10 marks)** | **(5 marks)**  The prototype brief provides a comprehensive and clear overview of the proposed solution in a concise paragraph, thoroughly outlining the prototype's requirements. Features of the prototype are detailed across 1-2 paragraphs, clearly describing the functionalities, components, and technical aspects. Potential limitations are identified with strong insight, demonstrating an understanding of possible challenges. Strategies for overcoming these limitations are creative and resourceful, clearly addressing each identified limitation. The work is logically structured and error-free, with professional presentation. | **(4 marks)**  The prototype brief offers a clear overview of the proposed solution, with only minor ambiguities. The features are described in detail, but some aspects might require additional explanation or could be less distinct. Potential limitations are addressed, though some gaps in identifying challenges exist. The strategies for overcoming limitations are clear but might lack creativity or thoroughness. The work is generally well-organised with few minor errors in presentation or clarity. | **(3 marks)**  The prototype brief is adequate, with a reasonable overview of the proposed solution, though some aspects might be vague or require more clarity. The features are outlined but might miss some crucial details. Potential limitations are identified, though the response lacks depth. The strategies to overcome these limitations are present but basic or underdeveloped. The work is organised, though there may be some structural issues or several grammatical errors. | **(2 marks)**  The prototype brief offers some overview of the proposed solution, but it is incomplete or unclear in several areas. The features are insufficiently described, with significant gaps or a lack of clarity. The section on potential limitations provides basic mention of challenges, but it lacks detail or relevance. The paragraph on overcoming limitations is minimal and provides little insight into resolving issues. The work is poorly organised with noticeable structural or grammatical errors. | **(1 marks)**  The prototype brief lacks a clear overview of the proposed solution, with significant confusion or misinterpretation. The features are either missing or described in a very elementary manner, with no clear indication of the key components. The section on limitations is largely absent, or challenges are not addressed meaningfully. Strategies to overcome limitations are either not considered or poorly explained. The work is disorganised and filled with significant grammatical or structural errors. |
|  | | | | | |
|  | | | | | |
| **Project Management – 10 marks** | | | | | |
| **Criteria** | **Extensive** | **Thorough** | **Sound** | **Basic** | **Elementary** |
| **Logbook**  **(5 marks)** | **(5 Marks)**  Detailed record of tasks completed throughout the duration of the project. Includes comprehensive reflections on tasks and solutions to problems encountered | **(4 marks)**  Clear record of tasks throughout the duration of the project. Includes thoughtful reflections on tasks and solutions to problems encountered. | **(3 marks)**  Records tasks completed during the duration of the project. Includes reflection on tasks and solutions to problems encountered. | **(2 marks)**  Recorded tasks completed are missing some portions of the project. Reflections on tasks and problems are basic. | **(1 mark)**  Record of tasks done during the project is incomplete. Little to no reflection on tasks and problems encountered. |
| **Gantt Chart**  **(5 marks)** | **(5 Marks)**  Gantt chart clearly details and expands on software report tasks to be completed. Initial and final time of tasks when completed are included. | **(4 marks)**  Gantt chart thoroughly details all software report tasks. Initial and final time of tasks are included. | **(3 marks)**  Gantt chart is sound. It is missing software report tasks and/or duration in the software development period. Initial and final time of completed tasks are included. | **(2 marks)**  Gantt chart is basic and hard to understand. It is missing software report tasks. Initial or final time of completed tasks are missing. | **(1 mark)**  Gantt chart is elementary and is missing most software report tasks. Does not track software report tasks on a timeline. |
|  | | | | | |
| **Prototype Development – 50 Marks** | | | | | |
| **Criteria** | **Extensive** | **Thorough** | **Sound** | **Basic** | **Elementary** |
| **Prototype/Code Completeness and Functional (20 marks)** | **(17 – 20 marks)**  The prototype with its codebase is complete, containing all the components and logic necessary for the prototype to function as intended. Every feature is implemented, and there are no missing parts. The code demonstrates a deep understanding of the project requirements, with all critical functions working flawlessly. Error handling and edge cases are accounted for, ensuring robustness and reliability. The code is clean, well-documented, and follows best practices, allowing others to understand and maintain it easily. | **(13 – 16 marks)x**  The prototype with its codebase is largely complete, with most features and functionality implemented. Minor components or secondary features might be missing, but the main functionality is present and works well. Error handling and edge cases are generally addressed, though some may require additional refinement. The code is mostly clean and follows best practices, with some areas needing further comments or documentation. Overall, the code functions as intended, with slight room for improvement in completeness. | **(9 – 12 marks)**  The prototype with its codebase provides an adequate implementation of the project, but some features or functionality might be incomplete or missing. The primary functions generally work, but secondary functions or edge cases could be lacking or not fully implemented. Error handling might be minimal or inconsistently applied. The code is somewhat clean but may require better organisation or additional comments for clarity. The project functions but might require significant revisions to ensure complete functionality. | **(5 – 8 marks)**  The prototype with its codebase is incomplete, with significant missing parts or features. The core functionality might be present, but several secondary functions or essential components are either not implemented or incorrectly coded. Error handling is minimal or non-existent, leading to potential crashes or unexpected behaviours. The code may be disorganised or lack proper comments, making it difficult to understand or maintain. The prototype partially functions but requires extensive work to be considered complete. | **(1 – 4 marks)**  The prototype with its codebase is highly incomplete, with most features or critical functions missing or incorrectly implemented. The code does not function as intended, with major components absent or failing. Error handling is entirely absent or improperly applied, leading to frequent errors or crashes. The code is disorganised and lacks any meaningful comments, making it challenging to understand or maintain. The prototype is far from functional and requires significant rework to be usable. |
| **Code Complexity**  **(5 marks)** | **(5 marks)**  The code demonstrates a balanced level of complexity, using appropriate structures such as functions, methods, or classes to achieve efficient and maintainable solutions. The project employs modularity and encapsulation, avoiding overly complex or monolithic code. It includes features that demonstrate a deep understanding of advanced programming concepts while maintaining readability and simplicity. The code architecture is well-designed, with clear separation of concerns and logical organisation, making it easy to understand and extend. | **(4 marks)**  The code generally uses functions, methods, or classes to achieve the project's goals, with some minor areas of inefficiency. It avoids excessive code in a single script and uses a modular approach, though some parts might be less organised or contain slightly complex constructs. The project includes some features that demonstrate advanced concepts, but it remains accessible and generally easy to follow. The overall structure is well-organised, but there might be a few areas where complexity could be reduced or improved. | **(3 marks)**  The code has a moderate level of complexity, with a mix of simple and complex structures. It generally uses functions and methods, but there might be parts where code is monolithic or lacks proper modularity. Some features might involve complex logic, but others may be overly simplistic or inefficient. The code structure is somewhat logical but could benefit from better separation of concerns or additional modularity. The project works, but there could be room for improvement in code complexity or efficiency. | **(2 marks)**  The code exhibits a basic level of complexity or disorganisation, with significant parts using monolithic scripts or redundant logic. It may lack appropriate use of functions, methods, or classes, leading to inefficient or overly complicated code. The project might have features that are either too complex or overly simplified, causing inconsistency in the code structure. The organisation is minimal, with limited separation of concerns, making the code hard to maintain or extend. | **(1 marks)**  The code is limited in complexity and disorganised, with most of the project contained in a single script or poorly structured. It lacks the use of functions, methods, or classes, leading to significant redundancy and inefficiency. The project might include complex logic with little to no modularity, making it extremely difficult to follow or maintain. The code structure is chaotic, with no clear separation of concerns, resulting in a project that is challenging to work with or extend. |
| **Code Efficiency**  **(5 marks)** | **(5 marks)**  The code is highly efficient, demonstrating a strong understanding of optimal algorithms and data structures. It uses the most performant methods to complete the project, with minimal redundancy and no unnecessary operations. The code is optimised for speed, memory usage, and resource management. It avoids excessive loops, redundant calculations, or costly operations, utilising best practices for efficiency. The project runs smoothly, even under stress or with large datasets, indicating a well-designed and optimised approach. | **(4 marks)**  The code is generally efficient, employing efficient algorithms and data structures in most areas. It uses performant methods to achieve project goals, though there might be some minor redundancies or inefficiencies. The code is optimised for speed and memory usage, but some parts might have slight room for improvement. The project operates efficiently but could be further optimised in certain sections to enhance performance. | **(3 marks)**  The code has a moderate level of efficiency, using basic algorithms and data structures. It generally performs well, but some areas might have redundant operations, excessive loops, or suboptimal logic. The code is mostly efficient, but some parts could be optimised to improve speed or memory usage. The project runs adequately, but may show some lag or inefficiency under heavy load or with large datasets. | **(2 marks)**  The code exhibits noticeable inefficiencies, with significant redundancy or suboptimal algorithms. It may use inefficient loops, redundant calculations, or unnecessary operations, leading to performance issues. The code is not well-optimised for speed or memory usage, and the project might experience noticeable lag or slow performance. The overall structure might contribute to inefficiency, with poor resource management or excessive complexity. | **(1 mark)**  The code is highly inefficient, with significant performance issues due to poor algorithm choices or redundant operations. It lacks optimisation for speed, memory usage, or resource management, leading to considerable lag or crashes under stress. The code may use highly inefficient loops, unnecessary calculations, or complex operations that greatly impact performance. The project struggles to run smoothly, indicating a lack of understanding of efficient coding practices. |
| **Links to the Planning**  **(5 marks)** | **(5 marks)**  The code is closely aligned with the planning and documentation outlined in previous project stages. It faithfully reflects the project's requirements, design specifications, and any preliminary planning or diagrams (such as context diagrams, flowcharts, or pseudocode – if applicable). All significant features and functionalities described in the documentation are implemented in the code, showing a strong connection between the design phase and the coding phase. The code structure and logic follow the documented plan, demonstrating a clear understanding of the project's goals and scope. | **(4 marks)**  The code generally aligns with the planning and documentation, with only minor deviations. It incorporates most features and functionalities outlined in the documentation, with some areas requiring additional alignment. The code largely follows the design specifications and reflects the initial plans, but a few elements might be implemented differently or need further clarification. Overall, the code demonstrates a strong link to the planning, but some aspects could be improved for complete alignment. | **(3 marks)**  The code has a moderate connection to the planning and documentation, but some significant elements might be missing or diverge from the initial plan. It reflects some of the features and functionalities described in the documentation, but there might be inconsistencies in how these are implemented. The code may contain unexpected changes or modifications not present in the original documentation, leading to some confusion about the intended scope. Despite these inconsistencies, the project generally aligns with the planning, but more effort is needed to achieve complete coherence. | **(2 marks)**  The code has a weak connection to the planning and documentation, with several significant elements missing or implemented incorrectly. The code deviates from the initial design specifications or preliminary planning, indicating a lack of alignment with the original project scope. The overall structure and logic might not reflect the planning phase, leading to a disjointed implementation. The code requires significant reworking to align with the earlier documentation. | **(1 mark)**  The code has little to no connection to the planning and documentation, with most features and functionalities not reflective of the initial design phase. The code may appear entirely different from the preliminary planning or diagrams, indicating a significant lack of coherence between the planning and coding stages. The structure and logic are disorganised, with no clear link to the project requirements or specifications. Overall, the code fails to align with the earlier documentation and requires extensive reworking to meet the original plan's goals and scope. |
| **Intrinsic Documentation**  **(5 marks)** | **(5 marks)**  The intrinsic documentation is thorough, providing comprehensive comments, inline explanations, and clear code annotations. Every significant section of the code is well-documented, with descriptive comments explaining the logic, purpose, and any critical decisions. Function and method headers include information on parameters, return types, and purpose. The code uses meaningful variable names and consistent naming conventions, making it easy to understand and maintain. Overall, the intrinsic documentation is exceptionally clear, enhancing readability and making the codebase accessible to other developers. | **(4 marks)**  The intrinsic documentation is generally thorough, with detailed comments and explanations throughout the code. Most significant sections are well-documented, but some areas might require additional comments or clearer annotations. Function and method headers are mostly complete, providing information on parameters and return types, but may have minor gaps. Variable names are generally meaningful, though there might be some inconsistencies in naming conventions. Overall, the intrinsic documentation is clear and useful, but a few areas could benefit from additional detail or refinement. | **(3 marks)**  The intrinsic documentation provides an adequate level of detail, with a reasonable number of comments and explanations. Most key sections have some level of documentation, but some parts might be vague or lack clarity. Function and method headers are present but might not cover all necessary information. Variable names are generally understandable, but there could be inconsistencies or less meaningful names in some areas. The intrinsic documentation is helpful but requires further refinement to improve clarity and consistency. | **(2 marks)**  The intrinsic documentation is minimal, with significant parts of the code lacking comments or explanations. The code may have some basic documentation, but many critical sections are left unexplained. Function and method headers might be missing or incomplete, providing little information on parameters or return types. Variable names may be unclear or follow inconsistent conventions, making the code harder to understand. Overall, the intrinsic documentation requires significant improvement to ensure the code is readable and maintainable. | **(1 mark)**  The intrinsic documentation is either non-existent or highly inadequate, with most of the codebase lacking comments or explanations. Critical sections of the code are undocumented, causing confusion and making the code challenging to understand. Function and method headers are either absent or incomplete, providing no useful information. Variable names are unclear or follow random conventions, leading to a chaotic codebase. Overall, the intrinsic documentation is insufficient, requiring major reworking to ensure the code is comprehensible and maintainable. |
| **Github Commits**  **(10 marks)** | **(9 – 10 marks)**  Commits are made regularly throughout the project, demonstrating consistent progress and a clear development timeline. Each commit is accompanied by a detailed and clear description, accurately reflecting the changes made. Version numbers are appropriately applied, following a logical sequence and adhering to proper versioning practices. The work shows excellent organisation, with no errors in the commit history or descriptions. | **(7 – 8 marks)**  Commits are made regularly with only minor gaps in consistency. Each commit includes a description, though some may lack the full clarity or detail necessary to fully capture the changes made. Version numbers are applied correctly, but there may be minor issues with sequence or versioning practices. The commit history is generally well-organised, with only minor errors. | **(5 – 6 marks)**  Commits are made with reasonable frequency but may have some inconsistencies, such as periods of inactivity or irregular updates. Descriptions for the commits are present but may be vague or lacking in necessary detail. Version numbers are used but might not always follow a consistent pattern. The commit history shows some organisation, though several errors or unclear entries are present. | **(3 – 4 marks)**  Commits are made sporadically, showing little evidence of a consistent workflow. Descriptions are brief and may be unclear or missing important details. Version numbers are inconsistently applied, and there are noticeable gaps in proper versioning practices. The commit history is disorganised and contains several errors or unclear entries. | **(1 – 2 marks)**  Few commits are made, showing a lack of engagement with the project. Descriptions are missing or overly vague, making it difficult to understand the changes made. Version numbers are either missing or improperly applied, with no adherence to versioning practices. The commit history is chaotic and full of errors, showing little to no organisation. |
|  | | | | | |
| **Evaluation – 10 Marks** | | | | | |
| **Criteria** | **Extensive** | **Thorough** | **Sound** | **Basic** | **Elementary** |
| **Evaluation**  **(10 marks)** | **(9 – 10 marks)**  The evaluation is comprehensive and critically reflective, thoroughly addressing all required elements. It provides a detailed recap of the project, offering insights into its development process and outcomes. The success or failure of the project is clearly explained, with compelling reasons for each. The evaluation thoroughly discusses how limitations were overcome, including detailed descriptions of solutions or adjustments made. It identifies and analyses the strengths and weaknesses of the project, highlighting key aspects of both. Problems encountered during the project are clearly explained, along with effective solutions and lessons learned. The evaluation concludes with a thoughtful discussion of what could be done differently or improved in future similar projects. | **(7 – 8 marks)**  The evaluation is detailed and reflective, covering most of the required elements, with minor gaps. It includes a solid recap of the project, with clear explanations of the success or failure and reasons behind it. It discusses how limitations were addressed, though some aspects might need further detail. Strengths and weaknesses are generally identified, but a few may require additional analysis. Problems encountered are mentioned, along with reasonable solutions, but some could be lacking in depth. The evaluation provides useful insights into what could be improved for future projects, with slight room for additional reflection. | **(5 – 6 marks)**  The evaluation offers an adequate reflection on the project, addressing most required elements. It provides a basic recap of the project, but the explanation of success or failure might be somewhat vague. The discussion on overcoming limitations is present but may be incomplete or lack specific details. Strengths and weaknesses are generally mentioned, but they could be explored further. Problems encountered are discussed, but solutions might be unclear or require additional explanation. The evaluation concludes with some thoughts on future improvements, but there's room for more critical analysis or insights. | **(3 – 4 marks)**  The evaluation has basic reflection, with significant gaps in addressing the required elements. The recap of the project is minimal or unclear, and the reasons for success or failure are either not well explained or missing. The discussion on overcoming limitations is basic, with vague or incomplete details. The strengths and weaknesses of the project are either not identified or briefly mentioned without much analysis. Problems encountered might be mentioned, but the solutions are unclear or ineffective. The evaluation provides minimal insights into what could be improved in future projects, lacking depth and critical reflection. | **(1 – 2 marks)**  The evaluation is highly inadequate, with most elements either missing or poorly addressed. The recap of the project is unclear or entirely absent, and there's no clear explanation of success or failure. The discussion on overcoming limitations is vague or absent, with no meaningful details. The strengths and weaknesses of the project are not identified, and problems encountered are either not mentioned or lack solutions. The evaluation provides no insights into what could be improved in future projects, offering little to no critical reflection or analysis. |

**TOTAL MARK /120**